НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

# Дипломна робота
**на здобуття ступеня бакалавра**

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Система поліпшення проведення занять (клієнт Android)

Виконав: студент 4 курсу, групи ТІ-51

Федьків Роман Романович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ходаківський Олександр Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний
Кафедра автоматизації проектування енергетичних процесів і систем
Рівень вищої освіти перший рівень
Напрям підготовки 6.050103 “Програмна інженерія’’

АТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2018 р.

**ЗАВДАННЯ**
**на дипломну роботу студенту**
Федьківу Роману Романовичу
(прізвище, ім’я, по батькові)

1.Тема роботи “ Система поліпшення проведення занять (клієнт Android) ”

керівник роботи доцент, к.т.н. Ходаківський Олександр Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”” 201 р.
№

2. Строк подання студентом роботи 201 р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Linux, мова програмування Kotlint, середовище розробки AndroidStudio .
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
1. Архітектура MVVM 2.Підключення локальної бази даних Room 3.Використання локальноъ бази даних 4. Підключення мобільної API - Firebase 5.Інтерфейс програмного продукту

Дата видачі завдання ”” 201 р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Вивчення та аналіз задачі | | |
| 2. | Розробка архітектури та загальної структури системи | | |
| 3. | Розробка структур окремих підсистем | | |
| 4. | Підготовка матеріалів | | |
| 5. | Програмна реалізація системи | | |
| 6. | Захист програмного продукту | | |
| 7. | Оформлення пояснювальної записки | | |
| 8. | Передзахист | | |
| 9. | Захист | | |

Студент _____ Федьків Р.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ходаківський О.В.
(підпис) (прізвище та ініціали)

## АНОТАЦІЯ

Мета роботи — поліпшення проведення занять та вирішення питання бар’єру між викладачами та студентами які навчаються в школах, вищих навчальних закладах. Для створення програмного продукту використовувалась середовище розробки Android Studio, мова програмування Kotlin та мобільне API — Firebase. Додаток має чисту архітектуру та структурний патерн MVVM, який гарантує розширювальну систему.
Записка містить 50 сторінок, 20 рисунків, 2 таблиці, 3 додатків і 18 посилань.
Ключові слова: ANDROIDSTUDIO, KOTLIN,АРХІТЕКТУРА MVVM, FIREBASE, API, JSON, БАЗА ДАНИХ, SQLITE, ROOM

## ABSTRACT

The purpose of the work is to improve the occupation and solve the barrier between teachers and students studying in schools and higher educational institutions. To create the software, the development environment of Android Studio, the Kotlin programming language and the mobile API - Firebase were used. The application has a clean architecture and structural MVVM pattern that guarantees an expansion system.
The note contains 50 pages, 20 figures, 2 tables, 3 annexes and 18 references.
Keywords: ANDROID STUDIO, KOTLIN, MVVM ARCHITECTURE, FIREBASE, API, JSON, DATABASE, SQLITE, ROOM

# ЗМІСТ

# ВСТУП

На сьогоднішній момент часу, в нашій країни та взагалі у всьому сучасному світі – не вистачає комп'ютерних систем, які б могли допомогти систематизувати проведення занять в університетах, школах та навіть на приватних курсах.

На даному етапі роботи було зроблено акцент на взаємодії студентів з викладачем під час лекцій. В майбутньому ставитиметься мета охопити повну взаємодію, тобто – виставлення балів, відмітки про присутність. Так як, зараз частіше за все лекції відбуваються у великих аудиторіях, де знаходиться безліч людей і тому у слухачів (наприклад - студентів) просто не має можливості поставити питання лектору. Саме таку проблему і розв'язує система написана мною.

Мобільні телефони вже давно стали невід'ємною частиною нашого життя. Наш телефон вже більше не є одним з пересічних пристроїв у нашому житті. Телефони стали нашими електронними біл-бордами, демонструючи хто і що для нас важливе. Після цього важко уявити, чому вони ще й досі не використовуються під час проведення занять та й взагалі усього учбового процесу.

Тому, було запропоновано дослідити, як саме можна організувати впровадження програмного продукту в процес проведення занять та реалізувати її. При чому реалізувати даний продукт як додаток для мобільного телефону, який використовує операційну систему Android.

Система, яка розробляється в теперішньому часі не повинна обмежуватись тільки для чогось одного. На мою думку, гаджети в теперішньому часі грають найбільшу роль з технічного прогресу, так як вони завжди з нами.

Мова програмування Kotlin - до початку 2017 року мова програмування Kotlin була відома тільки у вузьких колах розробників. Вона розроблена для роботи на віртуальній машині Java, але також може бути скомпільована в JavaScript і взаємодіяти безпосередньо з Java. Мова набула популярності в травні 2017 року, коли на конференції Google I/O 2017оголосили, що Kotlin буде включена у список офіційно підтримуваних мов для розробки Android-додатків.

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс . 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ARC — Automatic Reference Counter.

IDE — Integrated Development Environment.

DB— Database.

CLR — Common Language Runtime.

API — Application Programming Interface.

MVVM — Model-View-ViewModel.

UI - User Interface

SQL – Structured Query Language

Sqlite – Db for Android

Room – Db for Android

Firebase – API длямобільних додатків

JSON - JavaScript Object Notation

# 1. РОЗРОБКА ДОДАТКА ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID СПРЯМОВАНОГО НА СИСТЕМАТИЗАЦІЮ ПРОВЕДЕННЯ ЗАНЯТЬ.

Я вважаю, що система яка буде розроблятись, або яку розробляв я, повинна бути розроблена для мобільних телефонів, так як зараз майже у всіх під рукою є цей гаджет. Мій вибір був на користь операційної системи Android, так як вона охоплює великий відсоток людей по всьому світу (по статистиці близько 75 %). Декілька плюсів операційної системи Android:

- Простір у виборі персоналізації. Кожен користувач може завантажити лаунчер на свій смак і вільно встановити його.
- Можливість мати більше одного аккаунта.
- Наявність функції розширення пам'яті шляхом додавання sd-карти.
- Вільне скачування файлів з будь-якого ресурсу.

## 1.1. Взаємодія застосунків

Прикладний програмний інтерфейс (API) - набір визначеньпідпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек. В створеному програмному продукті використовується мобільне API — Firebase.

Головні переваги:

Швидкість роботи. У пакеті розробника Firebase зібрані інтуїтивно зрозумілі API, які спрощують і прискорюють розробку якісних додатків. Також у вашому

розпорядженні всі необхідні інструменти для розширення користувача бази і підвищення доходів - вам залишається тільки вибрати відповідні для ваших цілей.
Готова інфраструктура. Вам не доведеться створювати складну інфраструктуру або працювати з декількома панелями управління. Замість цього ви зможете зосередитися на потребах користувачів.
Статистика. В основі Firebase лежить безкоштовний аналітичний інструмент, розроблений спеціально для мобільних пристроїв. Google Analytics для Firebase дозволяє отримувати дані про дії ваших користувачів і відразу ж вживати заходів за допомогою додаткових функцій.
Кросплатформеність. Firebase працює на будь-яких платформах завдяки пакетам розробника для Android, iOS, JavaScript і C ++. Ви також можете звертатися до Firebase, використовуючи серверні бібліотеки або REST API.
Масштабованість. Якщо ваш додаток стане популярним і навантаження на нього зросте, вам не доведеться міняти код сервера або залучати додаткові ресурси - Firebase зробить це за вас. Крім того, більшість функцій Firebase безкоштовні і залишаться такими незалежно від масштабу ваших проектів. Платних функцій чотири. У них передбачено безкоштовний пробний період і два тарифних плани.
Безкоштовна підтримка по електронній пошті. Крім того, команда Firebase і фахівці з розробки Google дадуть відповіді на ваші питання на ресурсах Stack Overflow і GitHub.

## 1.2. Призначення та користувачі

Програмне забезпечення створене мною призначене для систематизації проведення занять, здебільшого лекцій, в університетах, школах, курсах і т.д. За допомогою даного додатку, власники гаджетів на платформі Android зможуть мати змогу задавати питання викладачам Навіть якщо особа не присутні на занятті, вона матимете змогу побачити відповідь на поставлене Вами питання. Викладач може відповідати під час або після заняття, також самі учні можуть відповідати на поставлені питання, а викладач позначати їх відповіді як правильні.

Це вирішує питання бар'єру між викладачами та студентами, які навчаються в школах, вищих навчальних закладах, різного роду курсів.

## 1.3. Задачі, які розв’язуються програмним забезпеченням

Це вирішує питання бар'єру між викладачами та студентами, які навчаються в школах, вищих навчальних закладах, різного роду курсів. Також це буде сприяти кращому сприйняттю інформації. Головною задачею даного програмного забезпечення є допомогти поліпшити та частково автоматизувати процес проведення занять. Для цього потрібно зробити акцент на взаємодії студентів з викладачем.

Тому, я вважаю, що спочатку потрібно зробити акцент саме на проведенні лекційних занять. Зараз лекції проводяться у великих аудиторіях, де знаходиться безліч людей. Саме через це у слухача іноді просто не має можливості запитати лектора. Даний додаток буде розв’язувати саме цю проблему комунікації викладача з студентами під час занять.

## Висновки до розділу 1

Отже, у цьому розділі було розглянуто чому нинішні гаджети є найліпшим способом просування будь-якої системи. Також було описано потенційних користувачів та опис проблем, які вирішує даний додаток.

# 2. АНАЛІЗ ПРОБЛЕМИ ПРОВЕДЕННЯ ЗАНЯТЬ В УЧБОВИХ ЗАКЛАДАХ ТА АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ

З кожним роком все більшої популярності набирають мобільні телефони, планшети та інші гаджети, які можуть бути поряд з кожною людиною завжди. Також всі люди, в якому віці вони б не були постійно навчаються чомусь новому. З метою допомогти викладачам та слухачам при навчанні було розроблено універсальну платформу для поліпшення проведення занять у вищих навчальних закладах, школах, різноманітних курсах і т.д.

За допомогою цього продукту слухачі та викладачі зможуть мати більше часу щоб зрозуміти один одного.

Є багато продуктів схожого роду, але вони представляють з себе дуже важку та незрозумілу програму, яка не підходить для щоденного використання.

Також є пробеми з тим що мова програмування Java є не дуже стилізована, і це тягне за собою такі фактори, як важкий та довгий код. Багатослівність коду може показатися перевагою, що допоможе при вивченні мови. Проте, довгі, надзвичайно складні пропозиції заблокують чтение і перегляд кодів. Як і естественные языки, многие мови програмування високого рівня містять лишнюю інформацію. Java - це більш легка версія неприступного C ++, яка видається програмістами, що прописують свої слова з англійської мови. Це робить мову більш зрозумілою для неспеціалістів, але менш компактною.

Тому я зробив перевагу в сторону програмної мови Kotlin. Яка в свою чергу має такі плюси, як:

- Лаконічність мови. Kotlin дуже схожий на Swift. Свіфт дуже лаконічний мову з ідеологією, що код читається як текст англійською мовою.

- Розширення, касти, іменовані аргументи і безліч інших кілер-фіч мови
- Мова підтримується Google і вже вийшло кілька стабільних версій
- Повна сумісність з Java. Класи на Котлін можна використовувати в Джаві і навпаки і все це в одному проекті
- Котлін це не академічний мова, який придумали в науковому колі для специфічних завдань. Це індустріальна мова, створена розробниками спеціально для розробки додатків Android

Для того, щоб підключити Kotlin не потрібно все переписувати. Можна робити тільки нові частини на Kotlin.

## 2.1. Аналіз існуючих програмних засобів

На платформі Android є декілька аналогічних систем, такі як - “Дневник.ру” або “Мой дневник”. В цих додатках є ряд недоліків:

- додатки спрямовані тільки на використання в школах
- громіздкий та незрозумілий функціонал
- не розширювальна система для інших навчальних закладів

В звязку з цим було прийняте рішення про написання нового програмного продукту.

На відміну від них, програмний продукт, який уже створений мною дуже зручний, і в ньому можна розібратись за лічені хвилини. Також його можна застосувати в усіх сферах навчання, школи, вищі навчальні заклади, курси. Також великим плюсом є лайв питання та відповіді на поточну лекцію.

Також в програмному продукті є два типи користувачів — це студенти та викладачі. На відміну від викладачів, учні мають змогу реєструватись самі, так як в них немає додаткового функціоналу для них. У свою чергу вчителів має реєструвати адміністратор, це зроблено для того щоб зменшити вірогідність взлому програми.

Додатковим функціоналом вважається відповіді, на які два відповідь сам вчитель або відповіді учнів, які викладач помітив як правильну, це сприяє ліпшому навчанню та більшій взаємодії між студентами та вчителі під час навчального процесу в школах, вищих начальних закладах та курсах.

## Висновки до розділу 2

У даному розділу було описано проблеми, які виникають при навчанні у школах, вищих навчальних закладах. Було описано обрану мову, її плюси та мінуси. Та порівняння з іншими програмними засобами, які зараз існуюсть на ринку.

# 3. ЗАСОБИ РОЗРОБКИ

Розробка Android-додатку потребує великих потреб зі сторони ПК(ноутбука), підходять операційні системи, такі як Windows, Ubuntu, Mac OS X. Я обрав Ubuntu так як вона є менш затратною.

Розробка програм під Android є достатньо затратним і звичайні комп'ютери не справляться з даним завданням. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto), також в ньому є емулятори на будь який смак.

Для розробки даного програмного продукту використовувався ноутбук від компанії "Acer" – "Swift 3" з встановленою останньою версією операційної системи Ubuntu 18.04.1 LTS. Даний комп'ютер було обрано, тому що розробка додатків для платформи Android вимагає хоча б 8Гб RAM.В якості мови було вибрано мову програмування Kotlin

## 3.1. Середовище розробки Android Studio

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс . 8 грудня2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0

Деякі особливості будуть пізніше розгорнуті для користувачів так як програмне забезпечення розвивається; наразі, передбачені такі функції[2]:

- Консоль розробника: підказки по оптимізації, допомога по перекладу, стеження за напрямком, агітації та акції — метрики Google аналітики.
- Резерви бета релізів та покрокові релізи.
- Базування на Gradle.

— Android-орієнтований рефакторинг та швидкі виправлення.
— Lint утиліти для охоплення продуктивності, юзабіліті, сумісності версій та інших проблем.
— Використання можливостей ProGuard та підписів до програм.
— Шаблони для створення поширених Android дизайнів та компонентів.
— Багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag-and-drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів.

### 3.1.1. Support Library

Support Library[11] - бібліотека, яка на старих версіях Android робить доступними можливості нових версій. Наприклад, фрагменти з'явилися тільки в третій версії (API Level 11). Якщо ви хочете використовувати їх у своєму додатку, цей додаток не буде працювати на старіших версіях Android, тому що ці старі версії ніколи не чули про клас android.app.Fragment[12]. Які тут є виходи?

- Додати в код перевірку версії системи і залежно від результату виконувати той чи інший код. Тобто якщо версія 11 і вище, використовуємо фрагменти, інакше Activity. Цілком реально, але не зовсім просто. Можна помилитися і заплутатися. Тобто Під час запуску програми на старих версіях доводиться або відмовлятися від нововведень і користуватися тим, що є, або винаходити велосипед і реалізовувати нововведення самому.
- Можна забити на старі версії і позиціонувати свій додаток тільки для нових версій. Тоді втрачається відчутна частина потенційних користувачів вашої програми. На момент написання цього матеріалу на версії Android нижче третьої сидить 69,7% користувачів. Відчутна така втрата вийде - більше, ніж дві третини! Звичайно, з часом все перейдуть на третю і наступні версії, і зможуть використовувати вашу програму. Але на той час вийдуть нові версії Android з новими можливостями, ви їх реалізуєте в своєму додатку і, тим

самим, знову відсіється частина користувачів. Загалом, вимальовується постійна дискримінація користувачів за версією.

– Використовувати бібліотеку Support Library. Вона містить класи - аналоги нововведень останніх версій, які будуть працювати на старих версіях.

На даний момент є дві бібліотеки v4 і v13. Цифра тут вказує мінімальний API Level на якому можна використовувати цю бібліотеку. Тобто додаток, що використовує v4, може бути запущено на API Level > = 4 і йому будуть доступні нововведення, які входять в цю бібліотеку (наприклад, фрагменти).

Бібліотеки ці періодично оновлюються, в них додаються нові класи, що реалізують нові можливості. Так що, якщо немає потрібних рішень, цілком можливо, що це з'явиться в майбутньому.

Отже, завдяки бібліотеці, один і той же код працює на старих і нових версіях і використовує можливості нових версій.

## 3.2. Мова програмування Kotlin

Kotlin[6] — статично типізованамова програмування, що працює поверх JVM і розробляється компанією JetBrains. Також компілюється в JavaScript. Мову названо на честь острова Котлін у Фінській затоці

Автори ставили перед собою ціль створити лаконічнішу та типо-безпечнішу мову, ніж Java, і простішу, ніж Scala. Наслідками спрощення, порівняно з Scala стали також швидша компіляція та краща підтримка IDE.

З 17 травня 2017 року входить в список офіційно підтримуваних мов для розробки додатків для платформи Android.

З 7 травня 2019 року є рекомендованою мовою для розробки Android додатків.

Він має підвищену продуктивність: програмний код на ньому виходить в середньому на 40% коротше, ніж на інших мовах, а також Kotlin дозволяє не допускати деякі помилки в коді. Одним з визначальних чинників популярності Kotlin в Google стало те, що він сумісний з Java, який вже використовується при розробці додатків під Android.

Плюси Kotlin:

— Шаблони рядків

— Сінглтон

— Null безпеку

— Функції розширення

— Розумні приведення типів (smart casts).

### 3.2.1. Null безпека в Kotlin

Система типів в мові Kotlin націлена на те, щоб викорінити небезпека звернення до null значенням, більш відому як "Помилка на мільйон".

Найпоширенішим підводним каменем багатьох мов програмування, в тому числі Java, є спроба зробити доступ до null значенням. Це призводить до помилки. В Java така помилка називається NullPointerException (скор. "NPE").

Kotlin покликаний виключити помилки подібного роду з нашого коду. NPE можу виникати тільки в разі:

— Явної вказівки throwNullPointerException ()

— Використання оператора !!

— Цю помилку викликав зовнішній Java-код

— Є якесь невідповідність при ініціалізації даних (в конструкторі використана посилання this на дані, що не були ще проініціалізовані)

Система типів Kotlin розрізняє посилання на ті, які можуть мати значення null (nullable посилання) і ті, які такими бути не можуть (non-null посилання). Наприклад, змінна часто використовуваного типу String не може бути null.

## 3.3. Firebase

Firebase — це платформи розробки мобільних та веб- застосунків. Firebase розвивається з 2011 року компанією Firebase Inc., яка була придбана Google у 2014.

Можливості Firebase:

- тепер це вже ціла платформа для побудови Android-, iOS- і мобільних веб-додатків, а не просто база даних в хмарі.
- база даних дозволяє працювати з даними, які зберігаються як JSON, синхронізуються в реальному часі і доступні за відсутності інтернету.
- firebase підтримує аутентифікацію по зв'язці електропошта + пароль, Facebook, Twitter, GitHub, Google та інші аутентифікаційні системи.
- крім бази даних Firebase пропонує хостинг статичних файлів для веб-сайту.

### 3.3.1. Firebase Auth

Firebase Auth — це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Він підтримує соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase.

### 3.3.2. Realtime Database

Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Компанія також надає клієнтськібібліотеки, які дозволяють інтеграцію із застосунками Android, iOS, Java, Objective-C, Swift. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері.

## 3.4. Room

Room[4] - це новий спосіб зберегти дані додатків в Android-додатку. Це частина нової Android Architecture, група бібліотек від Google, які підтримують доречну архітектуру додатків. Room пропонується в якості альтернативи Realm, ORMLite, GreenDao і багатьом іншим.

Room[5] - це високорівнева інтерфейс для низькорівневих прив'язок SQLite, вбудованих в Android, про які можна дізнатися більше в документації. Він виконує велику частину своєї роботи під час компіляції, створюючи API-інтерфейс поверх вбудованого SQLite API, тому вам не потрібно працювати з Cursor або ContentResolver. Room має три основних компонента: Entity, Dao і Database.

```
package com.example.mydiploma.cache.dao

import ...

@Dao
interface SubjectCacheDao {

    companion object {
        const val TABLE_NAME = "subject_table"
    }

    @Insert
    fun saveSubject( subjects : List<SubjectEntity>)

    @Query( value: "SELECT * FROM $TABLE_NAME")
    fun getSubjects() : LiveData<List<SubjectEntity>>

    @Query( value: "DELETE  FROM $TABLE_NAME")
    fun deleteAll()
}
```

Рисунок 3.1 - інтерфейс Dao

Анотацією Entity нам необхідно позначити об'єкт, який ми хочемо зберігати в базі даних.

В об'єкті Dao ми будемо описувати методи для роботи з базовими даними(рисунок 3.1).

Анотацією Database помічаємо основний клас по роботі з базою даних. Цей клас повинен бути абстрактним і успадковувати RoomDatabase. В параметрах анотації Database вказуємо, які Entity будуть використовуватися, і версію бази. Для кожного Entity класу зі списку entities буде створена табліца. В Database класі необхідно описати абстрактні методи для отримання Dao об'єктів, які вам знадобляться.

### 3.4.1 Type Converters та JSON

Іноді Entity об'єкти можуть містити поля, які не є примітивами, і не можуть бути збережені в БД. Також до цієї категорії перепадають списки або лісти (List, Array і т.д.)

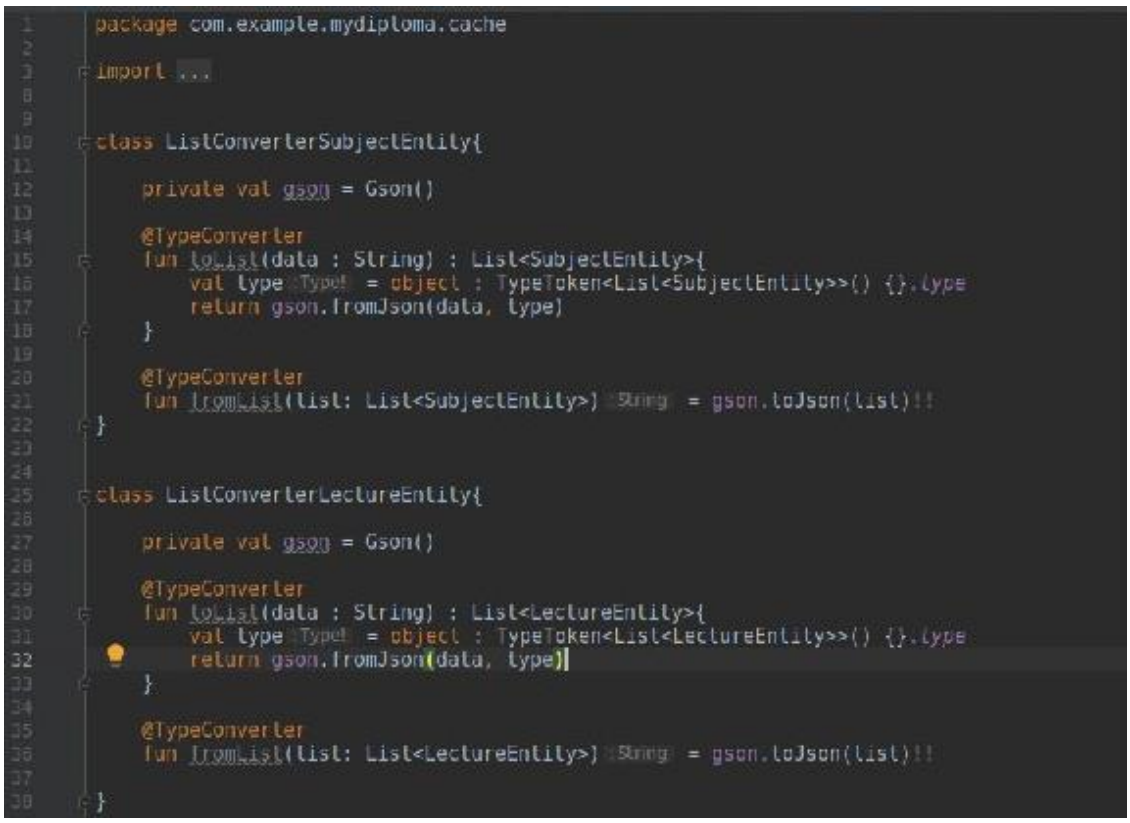
Зараз буде показано, як легко можна використовувати Type Converter для зберігання будь-якого об'єкта в Room, швидко змінюючи користувальницьку POJO з офіційної документації Google, включивши ще одне поле List.

В якості приклада розглянемо клас предметів (рисунок 3.2). Ми не можемо зберігати клас лекцій в ньому, тому що Room не знає як це зробити.

```
1   package com.example.mydiploma.data.model
2
3   import ...
6
7   @Entity(tableName = SubjectCacheDao.TABLE_NAME)
8
9   data class SubjectEntity (
10      var description : String = "",
11      @PrimaryKey
12      var id : String = "",
13      var name : String = "",
14      var teacher : String = ""
15  )
```

Рисунок 3.2 - клас предмету

Room справедливо зауважує, що поняття не має, як йому таке поле зберегти базу, і пропонує використовувати type converter (рисунок 3.2).



```
package com.example.mydiploma.cache

import ...


class ListConverterSubjectEntity{

    private val gson = Gson()

    @TypeConverter
    fun toList(data : String) : List<SubjectEntity>{
        val type Type! = object : TypeToken<List<SubjectEntity>>() {}.type
        return gson.fromJson(data, type)
    }

    @TypeConverter
    fun fromList(list: List<SubjectEntity>) String = gson.toJson(list)!!
}


class ListConverterLectureEntity{

    private val gson = Gson()

    @TypeConverter
    fun toList(data : String) : List<LectureEntity>{
        val type Type! = object : TypeToken<List<LectureEntity>>() {}.type
        return gson.fromJson(data, type)
    }

    @TypeConverter
    fun fromList(list: List<LectureEntity>) String = gson.toJson(list)!!

}
```

Рисунок 3.3 - конвертер для бази данних

Конвертер також можна вказати для всього об'єкта об'єкта. Це може бути корисно, якщо у в Entity кілька полей вимагають конвертери. Ви створюєте один клас, там прописуєте всі необхідні методи перетворення поля, і вказуєте цей клас для всього Entity.

```
package com.example.mydiploma.cache

import ...

@Database(entities = [LectureEntity::class, SubjectEntity::class],version = 1)
@TypeConverters(ListConverterLectureEntity::class, ListConverterSubjectEntity::class)
abstract class DbAbstract : RoomDatabase() {

    abstract fun lectureCacheDao() : LectureCacheDao

    abstract fun subjectCacheDao() : SubjectCacheDao

}
```

Рисунок 3.4 - абстрактний клас бази данних

Ну і найглобальне рішення - прописати конвертер для Database(рисунок 3.4) В цьому випадку Room зможе використовувати його у всіх Entity і Dao. Якщо у вас кілька конвертерів, вказуйте їх через кому.

Пару слів про JSON.

JSON — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).
Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

## 3.5 Coroutines in Kotlin

Coroutines спрощує асинхронне програмування, ховаючи всю складність всередині бібліотек.

Ця ідея зародилася в далекому 1967 році в мові програмування Simula. Спочатку методів було всього кілька: detach і resume (отже, вони дозволяли зупиняти і запускати виконання).

Проте, ідея була відкинута, коли з'явилися потоки. З одного боку, має сенс запускати асинхронну завдання в окремому потоці. Однак в деяких випадках, це рішення не буде ефективним. Наприклад, в таких[13]:

- Обидва потоки і перемикання між ними дорого обходяться;
- Створення тисяч потоків може істотно навантажити систему;

- Програмний код повинен використовувати тільки один потік (JavaScript, наприклад);
- Відсутність потреби використовувати потоки, тому що у вас багато «mutable state»;
- Можливість змінити призначений для користувача інтерфейс тільки з його потоку, тому легко помилитися.

По суті, coroutines - це обчислення, які можна призупинити, не блокуючи потік.

Є дві функції для запуску корутин[14]:

- launch {}
- async {}
- withContext()

Різниця в тому, що launch {} нічого не повертає, а async {} повертає екземпляр Deferred, в якому є функція await (), яка повертає результат корутіни, прямо як Future в Java, де ми робимо future.get () для отримання результату.

Також корутини не можна викликати в основному потоці подій, тому що буде викликана помилка:

- Функції переривання можна викликати тільки з корутіни або іншої функції переривання.

Функція затримки є функцією переривання, відповідно ми можемо її викликати тільки з корутіни або іншої функції переривання.

## Висновки до розділу 3

У даному розділі описано середовище програмування, мову програмування, використовувану базу даних та декілька сторонніх бібліотек. Також описано обгрунтування такого підходу до створення програмного продукту

# 4. ОПИС ПРЕДМЕТНОЇ РЕАЛІЗАЦІЇ

Система створена мною буде складатись з 3 основних модулів та 2 додаткових. Головним модулем, буде сторінка лайф, на якій можна задавати питання та отримувати відповідь в реальному часі. Інші два основні модулі - це модуль перегляду попередніх лекцій та модуль профілю. Два додаткових модулі це модуль входу та реєстрації в додатку.

## 4.1. Архітектура додатку

Архітектура програмного забезпечення — спосіб структуруванняпрограмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру

Проектування архітектури ПЗ — це процес розроблення, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи.

### 4.1.1. Архітектурний шаблон MVP

Model-View-Presenter (MVP) - шаблон проектування, похідний від MVC, який використовується в основному для побудови призначеного для користувача інтерфейсу (рисунок 4.1).

Елемент Presenter в даному шаблоні бере на себе функціональність посередника (аналогічно контролера в MVC) і відповідає за управління подіями призначеного для користувача інтерфейсу (наприклад, використання миші) так само, як в інших шаблонах зазвичай відповідає уявлення.

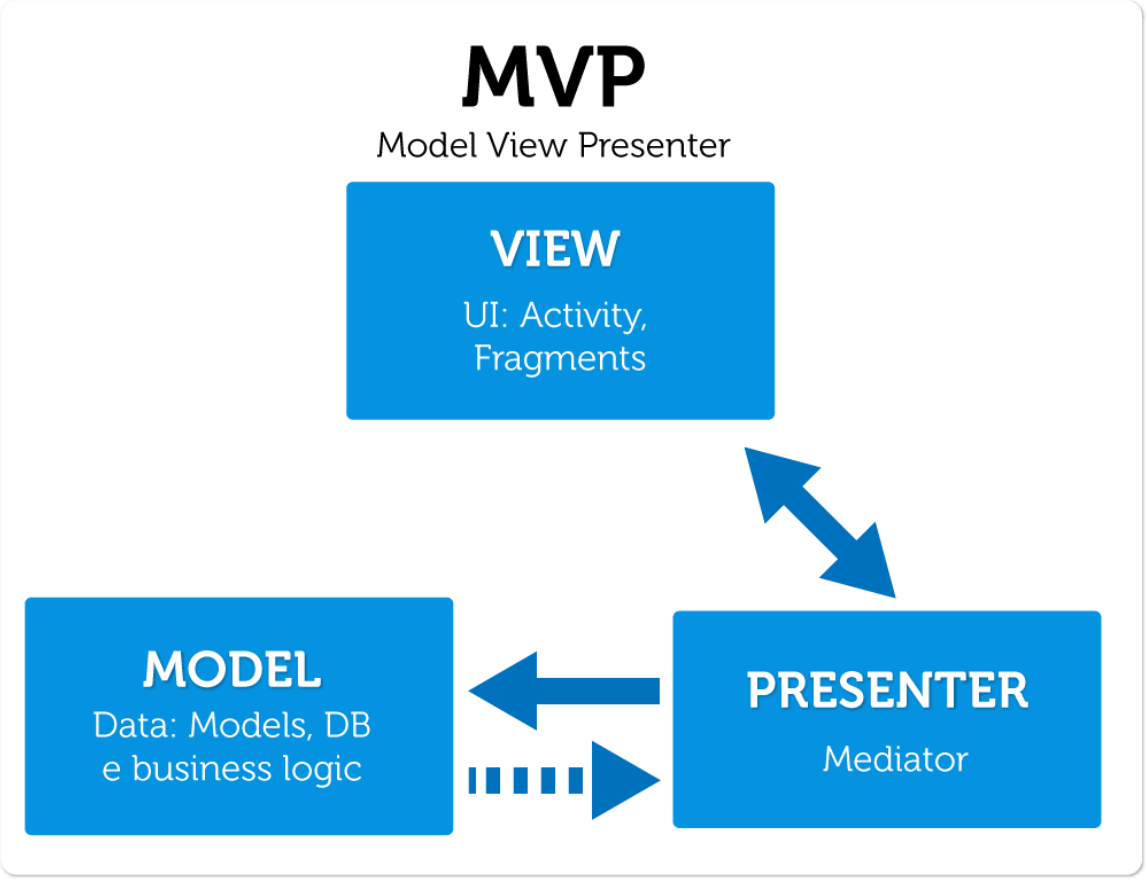



Рисунок 4.1 – Архітектура  MVP модуля

Якщо в Activity користувач натиснув кнопку Оновити, то Activity повідомляє про це презентери. При цьому Activity не вимагає презентер завантажити дані. Воно просто повідомляє, що користувач натиснув кнопку Оновити. А презентер вже сам вирішує, що після натискання цієї кнопки треба робити. Він запитує дані у моделі і передає їх в Activity, щоб відобразити на екрані.

Якщо екран відображає дані з бази даних, то модель - це база даних. Презентер може підписатися на повідомлення моделі про оновлення. У разі, коли дані в БД зміняться, модель сповістить про це презентер. Презентер отримає ці зміни і передасть їх в Activity.

Можна сказати, що презентер - це логіка, винесена з Activity в окремий клас. А Activity залишається для відображення даних і взаємодії з користувачем. Якщо ви вирішили зробити інше Activity для відображення даних, то вам вже не потрібно

буде переносити логіку в нове Activity, можна буде використовувати готовий Presenter. А якщо ви вирішили поміняти логіку, то вам не потрібно буде лізти в Activity і там, серед коду, який відповідає за відображення даних і взаємодія з користувачем, шукати логіку і міняти її. Ви змінюєте код в презентери.

### 4.1.2.Архітектурний шаблон MVVM

Патерн MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатки від візуальної частини (подання). Даний патерн є архітектурним, тобто він задає загальну архітектуру програми.

Даний патерн був представлений Джоном Госсманом в 2005 році як модифікація шаблону Presentation Model і був спочатку націлений на розробку додатків в WPF. І хоча зараз цей патерн вийшов за межі WPF і застосовується в самих різних технологіях, в тому числі при розробці під Android, iOS.

Розглянемо детальніше дану архітектуру (рисунок 4.1):

- View - цей шар містить компоненти UI і відповідає за код, керуючий компонентами користувальницького інтерфейсу (view), такий як ініціалізація дочірніх view, відображення progressbar'a, введення даних користувачем, обробку анімацій і т.п. Наприклад, такий код міститься в активують і фрагментах.
- ViewModel - об'єкти цього класу надають дані для компонентів UI. У нашому випадку View використовуватимуть LiveData для відстеження змін даних в ViewModel.
- Model - модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

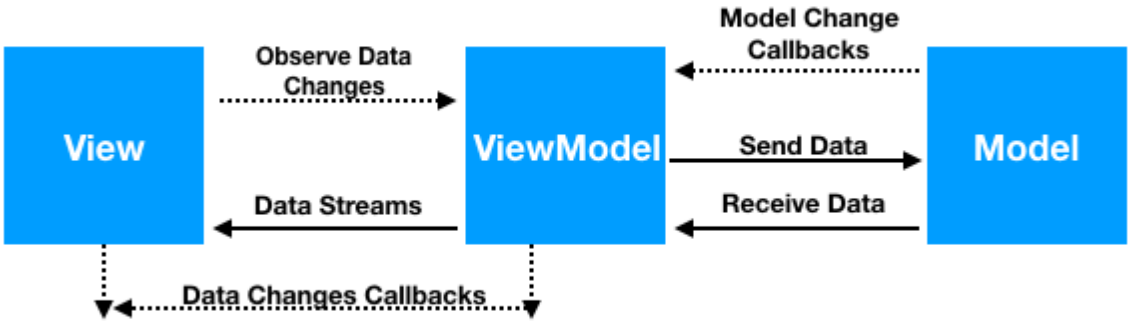



Рисунок 4.2 — Архітектура MVVM модуля

### 4.1.3. LiveData разом з MVVM

Компонент LiveData - призначений для зберігання об'єкта і дозволяє підписатися на його зміни. Ключовою особливістю є те, що компонент обізнаний про життєвий цикл і дозволяє не турбуватися про те, на якому етапі зараз знаходитися передплатник, в разі знищення передплатника, компонент відпише його від себе. Для того, щоб LiveData враховувала життєвий цикл використовується компонент Lifecycle, але також є можливість використовувати без прив'язки до життєвого циклу.

Клас LiveData, являє собою абстрактний дженерикових клас і інкапсулює всю логіку роботи компонента. Відповідно для створення нашого LiveData холдера, необхідно наслідувати цей клас, як типізації вказати тип, який ми плануємо в ньому зберігати, а також описати логіку поновлення зберігається об'єкта.

В ідеалі ViewModel не повинна нічого знати про Android. Це покращує тестованого і модульність, знижує кількість витоків пам'яті. Основне правило - у Вашій ViewModel не повинно бути імпорт android. * (За винятком  android.arch. *). Демонстрація MVVM модуля разом з LiveData[7] (рисунок 4.3).

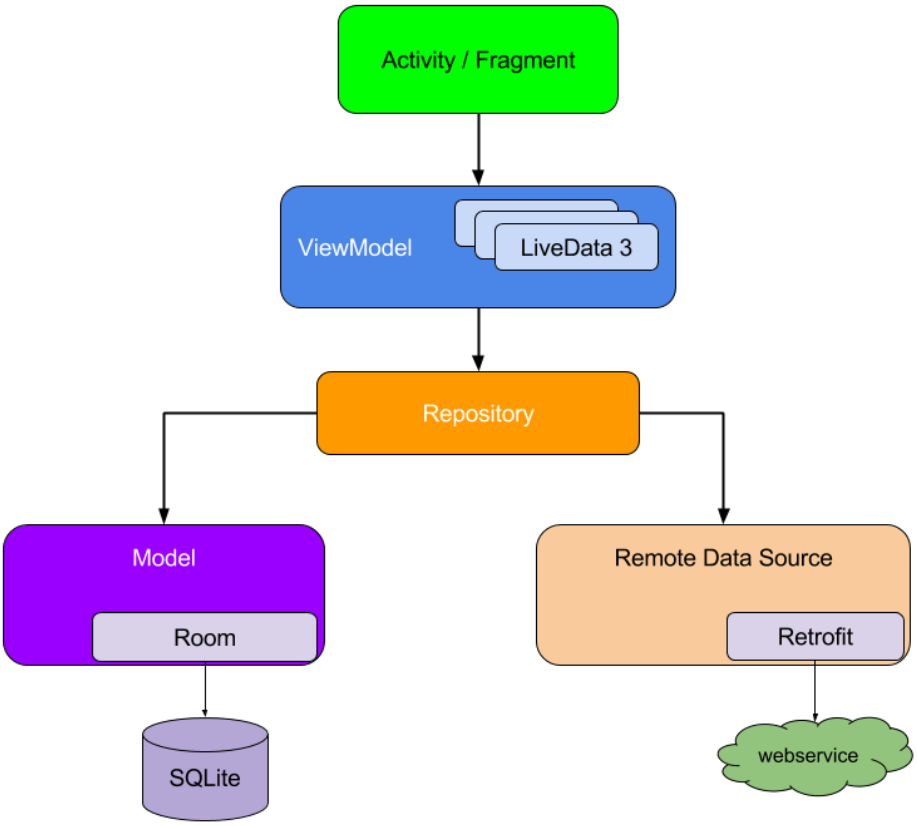



Рисунок 4.3 — Архітектура MVVM модуля разом з LiveData

Умовні оператори, цикли і спільні рішення повинні проводитися в ViewModel[8] або інших шарах додатки, не в активують або фрагментах. View зазвичай не піддається unit-тестування (крім випадків, коли використовується Robolectric), тому чим менше рядків коду - тим краще. View повинні тільки відображати дані і посилати дії користувача в ViewModel (або Presenter). Цей патерн називається Passive View. Скоуп ViewModels відрізняється від Скоуп активують або фрагмента. Тим часом, як ViewModel инициализирована і працює, активують може пройти через кілька станів життєвого циклу (рисунок 4.3)[1]. ViewModel може нічого не знати про те, що активують і фрагменти були знищені і створені.

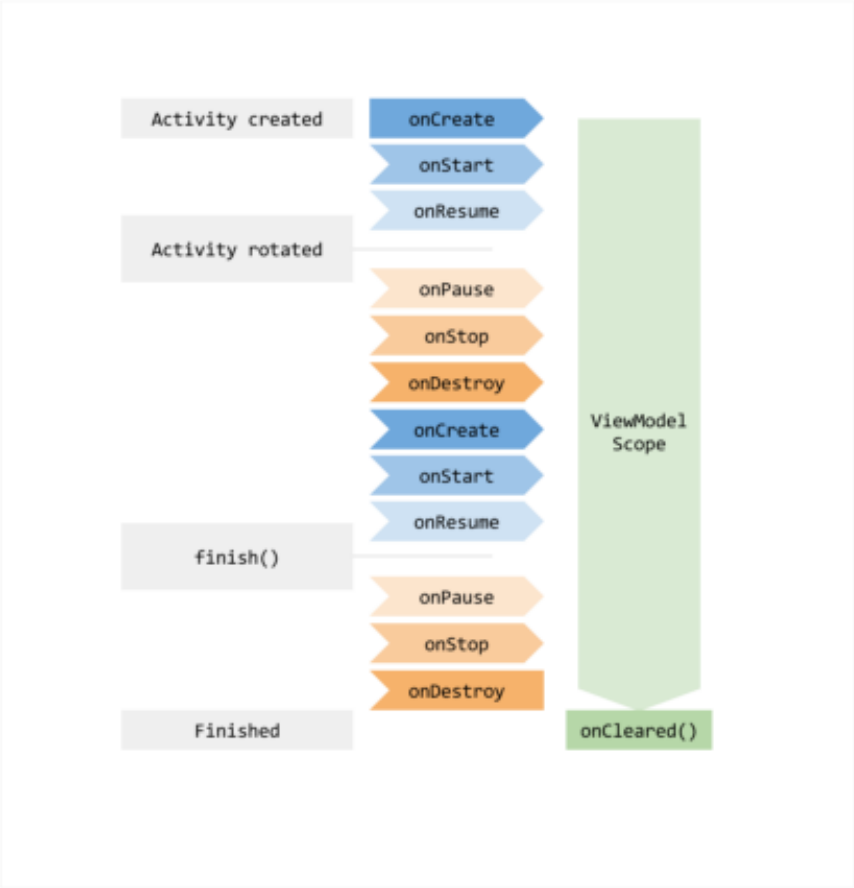



Рисунок 4.4 — Життєвий цикл ViewModel

### 4.1.4. Шаблон Singleton в Kotlin

Singleton - породжувальний шаблон проектування, що гарантує, що в однопроцесному додатку буде єдиний екземпляр деякого класу, і надає глобальну точку доступу до цього екземпляру.

Kotlin object[19] може впоратися з цим. Ключове слово companion object використовується для доступу до членів конкретного класу. Кожна змінна або метод розташовані всередині companion object можуть викликатися на ім'я класу (типу статичних).

## 4.2 Опис таблиць бази даних

У програмному продукті наявні дві таблиці, в яких зберігається інформація про предмети та лекції студента або викладача.

Room - база даних, яка використовувалась у додатку. Вона має абстрактний клас, та кількість об’єктів Dao, скільки моделей потрібно зберегти.

Нижче буде наведено їхню структуру

Таблиця 4.1 Структура таблиці “Предмет”

| **Ім’я поля** | **Тип і розмір поля** | **Опис поля** |
|---|---|---|
| Id | nvarchar(100) | Первинний ключ |
| Description | nvarchar(255) | Опис предмету |
| Name | nvarchar(100) | Назва предмету |
| Teacher | nvarchar(100) | Викладач |

Таблиця 4.2 Структура таблиці “Лекція”

| **Ім’я поля** | **Тип і розмір поля** | **Опис поля** |
|---|---|---|
| Id | nvarchar(100) | Первинний ключ |
| Description | nvarchar(255) | Опис лекції |
| SubjectId | nvarchar(100) | Id предмету |

Звязок між таблицями здійснюється через id об’єктів які ми зберігаємо в базу даних. В таблиці “Лекція” міститься поле subjectId, яке відповідає полю id в таблиці “Предмет”.

## 4.3 Узагальнення (Generics)

Узагальнене програмування - парадигма програмування, яка полягає в такому описі даних і алгоритмів, яке можна застосовувати до різних типів даних, не змінюючи саме це опис

### 4.3.1 Створення параметризованих класів

В програмному продукті використовуються 4 параметризовані класи, вони слугують для зменшення коду в самому проекті та кращої читабельності. Параметризований клас - Adapter (рисунок 4.4) є прикладом узагальнень.

```
class Adapter<T>(
    private val itemClickListener: (T) -> Unit

) : RecyclerView.Adapter<Holder<T>>() {

    private var list = mutableListOf<T>()

    fun updateList(list : List<T>) {
        this.list = list as MutableList<T>
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder<T> {
        val view :View! = LayoutInflater
            .from(parent.context)
            .inflate(R.layout.item_for_rec_view, parent, attachToRoot: false)
        return Holder(view,itemClickListener)
    }

    override fun getItemCount(): Int = list.size

    override fun onBindViewHolder(holder: Holder<T>, position: Int) {
        holder.bind(list[position], position)
    }
}
```

Рисунок 4.5 - Параметризованийадаптер

Впрограмномукласіє 3 методаякіперевизначаютьсятакякминаслідуємосьвідкласуадаптеру RecyclerView.Adapter<Holder<T>>. Також є метож для оновлення масиву з даними та нотифікації про нові дані.

```
viewL.recyclerView.layoutManager = LinearLayoutManager(context)
viewL.recyclerView.adapter = Adapter<SubjectEntity>{ it:SubjectEntity
    navigationInterface.goToLectureFragment(it.id, it.name)
}
```

Рисунок 4.6 - Використання параметризованого адаптера

Створюючи єкземпляр цього класу вказуємо який тип буде використовуватись (рисунок 4.5). Kotlin може вивести універсальний тип на тип параметра, тому можна опустити це при використанні конструктора.

### 4.3.2. Ключове слово *out*

Створюючи клас, який буде виробляти результат деякого типу T[15]. Іноді можемо привласнити це вироблене значення посиланню, яке має супертип типу T. Щоб домогтися цього за допомогою Kotlin, нам потрібно використовувати ключове слово out для універсального типу. Це означає, що ми можемо присвоїти це посилання будь-якого з її супертіпа. Вихідна значення може бути зроблено тільки даним класом, але не використано

### 4.3.3. Ключове слово in

Інколи маємо протилежну ситуацію, яка б означала, що є посилання типу T[16], і потрібно мати можливість призначити її підтипу T. Є можливість використовувати ключове слово in для універсального типу, якщо є потреба привласнити його посиланням на його підтип. Ключове слово in може використовуватися тільки для типу параметра, який використовується, але не створюється.

## 4.4 RecyclerView

Елемент RecyclerView - це компонент інтерфейсу користувача, який дозволяє нам створювати прокручуваний список.

Щоб мати змогу створити цей компонент, потрібно до залежностей додати бібліотеку, яка відповідає за цей елемент. Також потрібно подивитись на останні оновлення, щоб бути завжди в курсі останніх новин по цьому елементу View.

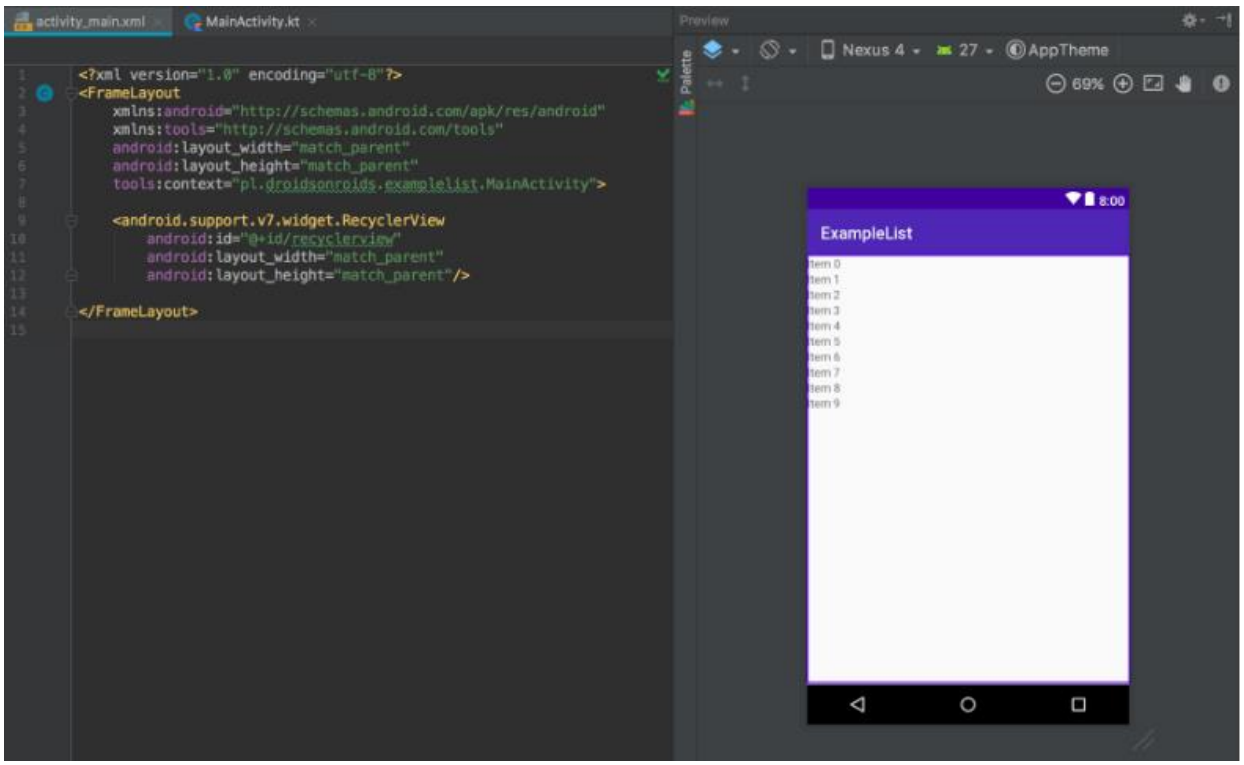



Рисунок 4.7 - RecyclerView

```
class MainActivity: AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val recyclerView: RecyclerView = findViewById(R.id.recyclerview)
        recyclerView.layoutManager = LinearLayoutManager( context: this)
    }

}
```

Рисунок 4.8 - Створення RecyclerView в Activity

На рисунку 4.6 зображено графічне представлення компонента RecyclerView, в основні характеристики цього елемента входить висота, ширина та id. Також після цього в обов’язковому порядку потрібно проініціалізувати цей елемент (рисунок 4.7). В Activity ми вказуємо LinearLayoutManager для компонента, він відповідає за позиціонування view-компонентів в RecyclerView, а також визначення того, коли слід перейти на перегляд-компоненти, які більше не використовуються.

Таким чином, це означає, що LayoutManager відображає список в певній формі. У цьому класі ми використовували LinearLayoutManager, який показує дані в простому списку - вертикальному або горизонтальному (за замовчуванням вертикальному). Але це можна дуже просто змінити, використавши інший LayoutManager. Наприклад GridLayoutManager, StaggeredGridLayoutManager або WearableLinearLayoutManager.

Також для кожного елемента списку можна використовувати або вже готовий елемент View або створити власний за допомогою елементів в Android Studio. У створеному програмному продукті використовуються прості View-компоненти. Так для відображення предметів було взято два текстових поля, одне з яких відповідає за номер предмету, а друге - за назву.

### 4.4.1. Adapter та ViewHolder для RecyclerView

Адаптер, який використовується для RecyclerView[17] обов’язково має перевизначити об’єкт типу ViewHolder, який дає змогу побачити кожен елемент представлення.

Повинні створити клас, який розширює RecyclerView.Adapter, який в якості параметра приймає клас, який розширює RecyclerView.ViewHolder. Також необхідно перевизначити деякі необхідні методи[18]. Що роблять ці методи?

- getItemCount () повертає загальна кількість елементів списку. Значення списку передаються за допомогою конструктора.
- onCreateViewHolder () створює новий об'єкт ViewHolder щоразу, коли RecyclerView потребує цього. Це той момент, коли створюється layout рядки списку, передається об'єкту ViewHolder, і кожен дочірній view-компонент може бути знайдений і збережений.
- onBindViewHolder () приймає об'єкт ViewHolder і встановлює необхідні дані для відповідного рядка у view-компоненті.

Існує методика для поліпшення продуктивності роботи великих списків - використання класу ViewHolder (рисунок 4.8). Метод findViewById () досить важкий

в плані споживання ресурсів, так що потрібно уникати його, якщо в ньому немає прямої необхідності. ViewHolder зберігає посилання на необхідні в елементі списку шаблони. Цей ViewHolder прикріплений до елементу методом setTag (). Кожен елемент списку може містити застосовану посилання. Якщо елемент очищений, ми можемо отримати ViewHolder через метод getTag ().

```
class Holder<T> (
    itemView : View,
    private val itemClickListener: (T) -> Unit
) : RecyclerView.ViewHolder(itemView) {

    fun bind(entity : T, position : Int): Unit = with(itemView) {
        if (entity is SubjectEntity){
            subjectName.text = entity.name
            subjectNumber.text = (position + 1).toString()
            subjectName.setOnClickListener { it:View!
                itemClickListener(entity)
            }
        }
        if (entity is LectureEntity){
            subjectName.text = entity.description
            subjectNumber.text = (position + 1).toString()
            subjectName.setOnClickListener { it:View!
                itemClickListener(entity)
            }
        }
    }
}
```

Рисунок 4.9 - ViewHolder для RecyclerView

В методі bind() виконується вся робота з певним елементом. Отримуючи сутність певного об’єкту ми присвоюємо текстовим поля значення полів цього об’єкта. Також тут визначенний itemClickListener() який відповідає за натискання по певному елементу, реалізація цього кліку визначена в головному Activity() до якого прив’язаний RecyclerView().

## 4.5 Впровадження залежностей (Dependency injection)

Впровадження залежності — шаблон проектування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію управління» для розв'язання (отримання) залежностей.

Впровадження залежності — це передача залежності (тобто, сервісу) залежному об'єкту (тобто, клієнту). Передавати залежності клієнту замість дозволити клієнту створити сервіс є фундаментальною вимогою до цього шаблону проектування.

Існує три найбільш поширені форми впровадження залежностей:

- впровадження в конструктор,
- впровадження у властивість,
- впровадження в метод.

В програмному продукті використовується бібліотека Koin для впровадження залежностей. Koin - це невелика бібліотека для написання впроваджень залежностей. Без проксі, кодогенераціі і інтроінспекціі (introspection). Працює як Service Locator. Використовує DSL і фичи мови Kotlin. Сама бібліотека має на увазі, що буде використовуватися в додатках написані на Kotlin, але можна і з Java.

Є декілька способів для впровадження залежностей[10]:

- applicationContext - Це лямбда для створення Koin модуля. Ця функція повертає модуль Koin і є початком кожного визначення компонента в Koin.
- factory - Надання залежності як фабричний компонент, тобто створює кожен раз новий екземпляр.
- single - Надання залежності як Singleton.
- bind - Додаткове зв'язування типу Kotlin для даного визначення компонента.
- get - Дозволяє компонентні залежності. Функція сама зрозуміє яка залежність потрібно для кожного класу.
- context - Обьявление логічного контексту.
- viewModel - Спеціальне надання залежності для ViewModel, знаходиться в окремому пакеті.

Порівняно з бібліотекою Dagger2[9], яку рекомендує використовувати Google, у Koin є плюси  порівняно з данною бібліотекою:

- koin не використовує анотації
- koin має додаткові розширення щоб використовувати її для viewModel

- dagger2 написаний на Java, Koin - на Kotlin
- не має кодогенерації

Мінуси Koin:

- dagger2 має підтримку від Google, а не від одного розробника у випадку Koin, також всі помилки ми можемо побачити
- Помилки можна побачити тільки після запуску в runtime, у dagger2 - під час компіляції

## Висновки до розділу 4

У даному розділі було описано архітектурний підхід до розробки програмного продукту. Також описані основні засоби розробки додатку. Також описані патерни, які використовувались у створенні програмного продукту.

# 5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблений програмний код працює на всіх гаджетах на платформі Android. Тому кожен користувач з цією платформою зможе коритуватись додатком.

## 5.1 Інструкція з використання програмного продукту

Натиснувши на ярлик з назвою LectureTime, користувач побаче вікно вітання (рисунок 5.1).

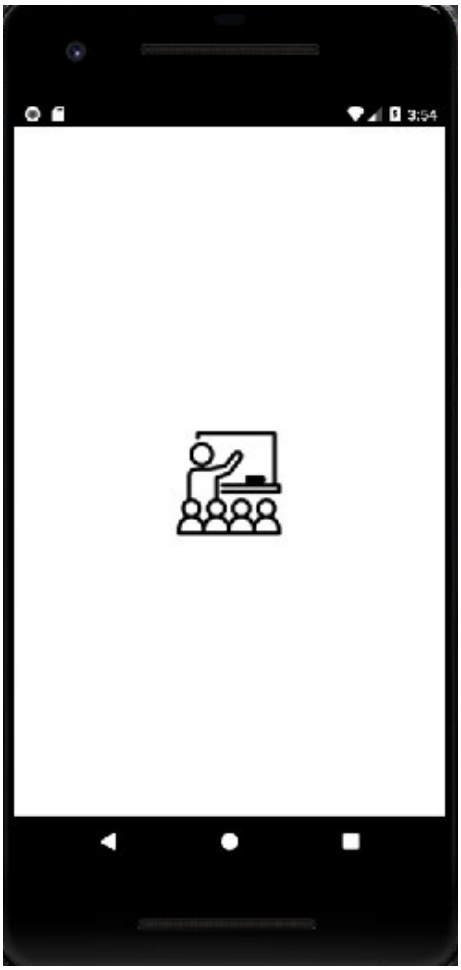


Рисунок 5.1 — Сторінка вітання

Після цього вітання, можна спостерігати на перехід до екрану входу (рисунок 5.2). На головному екрані входу потрібно ввести свій eMail та пароль, якщо користувач є

зараєстрованим, або свайпнути вправо та перейти на екран реєстрації (рисунок 5.3), то особа, яка хоче використовувати програмне забеспечення має заповнити такі поля як eMail, пароль, ім’я та прізвище після цього буде показаний прогрес бар який вказує на те, що йде завантаження користувача та його даних. Так як, цей програмний продукт є демонстраційною версією і розроблений тільки для групи ТІ-51, через че додаток передбачає авторизацію та реєстрацію студентів цієї групи або викладачів, які викладають в ній.

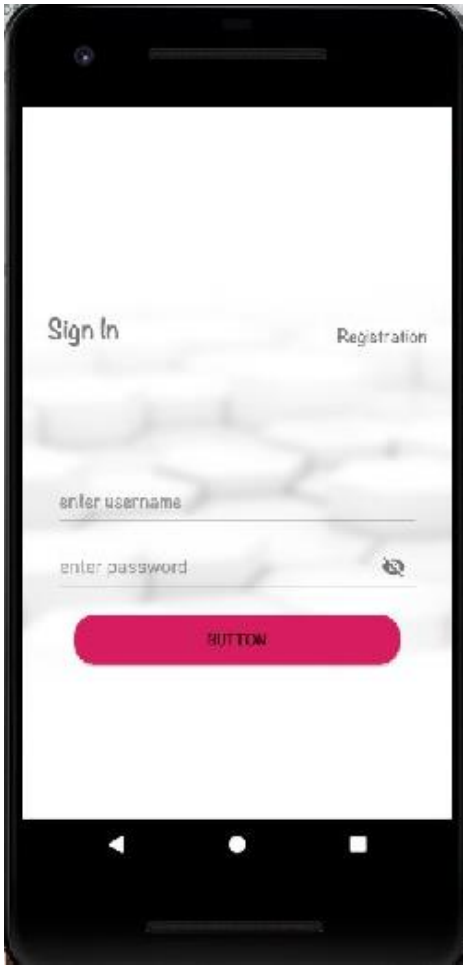



Рисунок 5.2 — Сторінка входу

Дуже важливо розуміти, що не пройшовши тап авторизації, користувач не зможе перейти до основних модулів програмного продукту. Це зроблено для того, щоб забезпечити безпечне користування даним додатком.

Якщо користувач ще не створив свій особистий акаунт, то він може це здійснити на екрані реєстрації (рисунок 5.3).

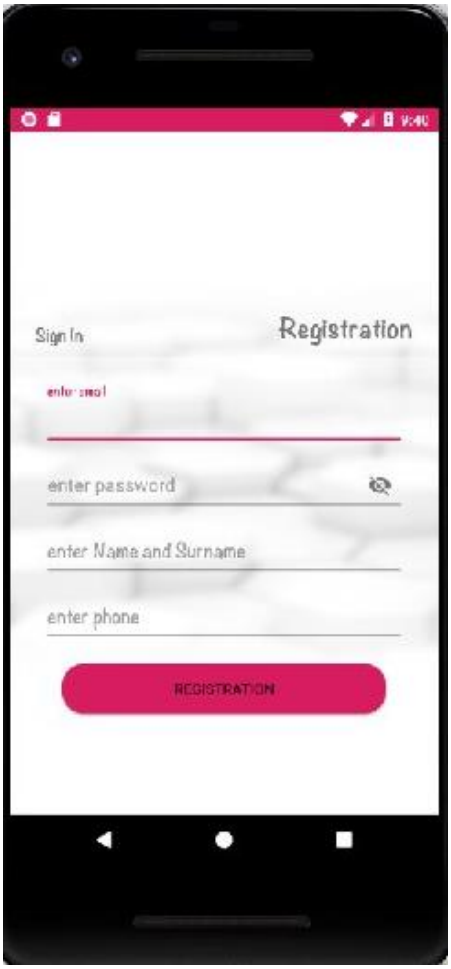



Рисунок 5.3 — Сторінка реєстрації

В головному вікні в нас є 3 головні сторінки. Перша це сторінка лайф (рисунок 5.4), в які вказується поточна пара та лекція. Тут можна задавати питання і отримувати відповіді в реальному часі. Набравши питання у відповідному полі та натиснувши клавішу Send буде відправлено запитання. Після цього є можливість натиснути на запитання, та перейти на вкладку відповіді, там інтерфейс ідентичний.

На цьому екрані користувач бачить всі питання щодо поточної лекції та також може задати питання, які йому цікаві. Всі питання є анонімними, що є досить важливим, так як студент не буде перейматись та ніяковіти, якщо він поставить очевидне запитання. Даний екран являється найважливішим в додатку, так як саме в ньому відбувається живий обіг питань під час лекції. Тобто студент під час лекції

просто має написати сюди запитання, і в нього не буде ніякої необхідності питати лектора наживо в аудиторії.


Рисунок 5.4 — Сторінка лайф

Друга це сторінка всіх предметів (рисунок 5.5), для вашої групи, з неї можна перейти до всіх лекцій, які відбулися до даного дня. Відповідно далі йдуть питання та відповіді від попередніх лекцій. Якщо наявне підключення до інтернету, то користувач отримає нові дані, але якщо інтернет буде відсутнім, то користувач отримає ті дані, які були завантажені минулого раза. Цю можливість нам забезпечує база даних Room, яка була встановлена в середовищі розробки Android Studio як стороння бібліотека, яка працює поверх вбудованої в середовище бази даних SqlLite.

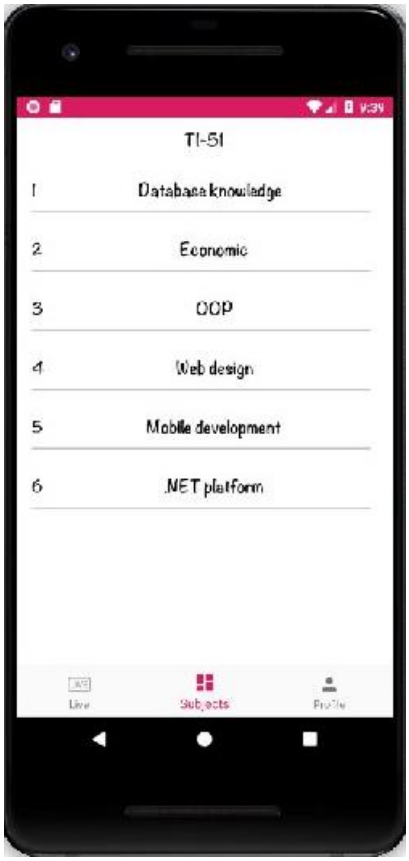



Рисунок 5.5 - Сторінка предметів

Ця сторінка дещо відрізняється для користувачів групи “Викладач” та “Студент”. Це обумовлено тим, що студенти можуть тільки переглядати відповіді дані на поставлене запитання. Викладач же в свою чергу, крім того що має перевагу в відповіді, його відповідь одразу помічається як вірна. Він має змогу ще й відмічати відповідь, як вірну. В майбутньому планується додати функціонал для змінення відповіді для користувачів групи “Викладач” та “Учнів”. Третя сторінка це сторінка поточного профілю користувача (рисунок 5.6), тут можна побачити інформацію про користувача, також присутня кнопка about. Кнопка exit присутня для виходу або зміни користувача в додатку.

На цій сторінці знаходиться інформація та деякі дані користувача, які було заповнено під час реєстрації. Також доступні дві дії – вихід з акаунта та перегляд сторінки “About”. Якщо користувач натисне кнопку “Exit” він буде перенаправлений на екран авторизації.


Рисунок 5.6 - Сторінка профілю

Сторінка “About” (рисунок 5.7) містить в собі іконку та назву даної системи, як і заставка при завантаженні додатка. Головними тезами цього екрану являється те, щоб донести до користувачів, що саме вирішує система і що саме планується зробити в майбутньому для того, щоб покращити та надати змогу автоматизувати процес проведення занять в університетах, школах та курсах.

Також ставиться за мету оповістити користувачів, що дана версія додатку поки що є демонстраційною, тобто вся функціональність та всі дані використовуються тільки одною групою ТІ – 51.

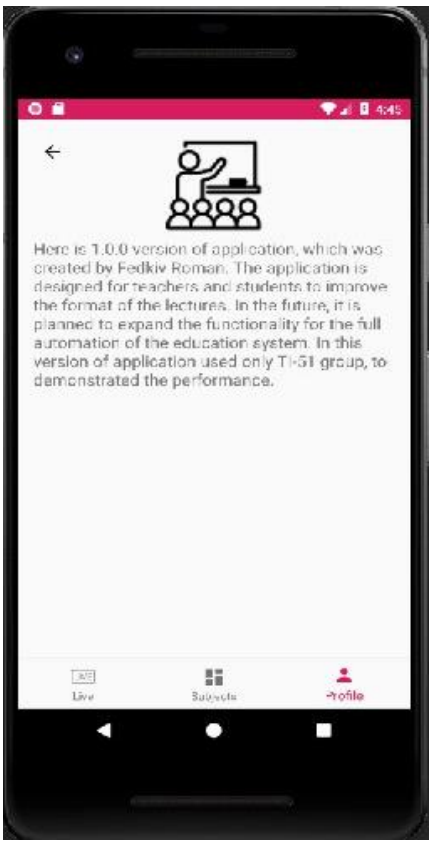



Рисунок 5.7 - Сторінка about

## Висновки до розділу 5

У даному розділі описано детальну взаємодію користувачів з програмним продуктом. Також описані основні сторінки додатку з рисунками та їх головний функціонал.

# ВИСНОВКИ

У ході розробки програмного продукту поліпшення проведення занять було знайдено та досліджено системи, які вирішують це поставлену задачу. Знайдені додатки, після глибокого аналізу, повністю не вирішують поставлену задачу. Тому були знайдені шляхи за допомогою яких буде покращено систему, яка розроблялась. Також у створених систем були такі недоліки, як високі вимоги до гаджетів на платформі Android.

Розроблений додаток дозволяє автоматизувати та поліпшити проведення занять у школах, вищих навчальних закладах, різного роду курсів і т.д.

Проаналізувавши методи вирішення цієї задачі було обрано таку стратегію. Використати веб-ресурс, мобільну Api – Firebase, для зберігання інформації про студентів, викладачів, предмети, лекції, питання та відповіді. Також було прийняте рішення про використання чистої архітектури, та структурного патерну MVVM для того щоб в подальшому розширювати створену систему.

У програмному додатку використовуються найновіші способи підходу до розробки програмного додатку, що є великим плюсом.

Після виконання тестових задач, було підтверджено правильність роботи системи.

Системою можуть користуватись студенти вищих навчальних закладів, шкіл або ж слухачі різноманітних курсів, також викладачі. Програмний продукт можуть використовувати всі в кого є гаджет під системою Android у вільному доступі з підключенням до інтернету. Також наявні модулі які можна використовувати без доступу до інтернету.

Реєстрація є тільки для студентів, так як це запобігає реєстрації користувачів як вчителя, та запобігає користування додатковим функціоналом доступним тільки для викладачів.

Тому, можна зробити висновок що диплом покращив знання різних технологій для розробки додатків на платформі Android, що розроблялась на мові Kotlin, в середовищі розробки Android Studio.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Роберт Мартин «Чистая архитектура. Искусство разработки программного обеспечения». — 2018. — Режим доступу: https://habr.com/ru/company/piter/blog/353170/
2. Билл Филлипс, Крис Стюарт, Кристин Марсикано. Android. Программирование для профессионалов. 3-е издание — 2016. — 688 с.
3. Room Persistence Library [Електроний ресурс] — Режим доступу до ресурсу: https://developer.android.com/topic/libraries/architecture/room
4. Room: Хранение данных на Android для всех и каждого [Електроний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/post/336196/
5. Дмитрий Жемеров, Светалана Исакова. Котлин в действии. — 2017. — 360с
6. LiveData Clean Code using MVVM and Android Architecture Components [Електроний ресурс] — Режим доступу до ресурсу: https://android.jlelse.eu/lets-keep-activity-dumb-using-livedata-53468ed0dc1f
7. MVVM Architecture & LiveData, ViewModel, LifeCycle Components [Електроний ресурс] — Режим доступу до ресурсу: https://android.jlelse.eu/android-architecture-pattern-components-mvvm-livedata-viewmodel-lifecycle-544e84e85177
8. Не Dagger'ом едины [Електроний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/post/352352/
9. Dependency Injection With Koin [Електроний ресурс] — Режим доступу до ресурсу: https://www.raywenderlich.com/9457-dependency-injection-with-koin
10. Ян Клифтон. Проектирование пользовательского интерфейса Android. — 2017. — 452с
11. Билл Филлипс, Билл Харди. Android. Программирование для профессионалов. 2-е издание — 2016. — 644 с.

12. Coroutines for Android [Електроний ресурс] — Режим доступу до ресурсу: https://proandroiddev.com/coroutines-for-android-6f9b9f966056
13. Improve app performance with Kotlin coroutines [Електроний ресурс] — Режим доступу до ресурсу: https://developer.android.com/kotlin/coroutines
14. Understanding Generics and Variance in Kotlin [Електроний ресурс] — Режим доступу до ресурсу: https://proandroiddev.com/understanding-generics-and-variance-in-kotlin-714c14564c47
15. Обобщения (Generics) [Електроний ресурс] — Режим доступу до ресурсу: https://kotlinlang.ru/docs/reference/generics.html
16. Using the RecyclerView[Електроний ресурс] — Режим доступу до ресурсу: https://guides.codepath.com/android/using-the-recyclerview
17. Производительность для списков - convertView, ViewHolder [Електроний ресурс] — Режим доступу до ресурсу: http://developer.alexanderklimov.ru/android/listview/listview-perfomance.php
18. Kotlin singletons with argument [Електроний ресурс] — Режим доступу до ресурсу: https://medium.com/@BladeCoder/kotlin-singletons-with-argument-194ef06edd9e

# ДОДАТОК А

Система поліпшення проведення занять (клієнт  Android)

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51194_19Б

Аркушів 2

Київ - 2019

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51194_19Б | Записка_Федьків.docx | Текстова частина дипломної роботи |
| Компоненти | | |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51194_19Б | Diploma.zip | Основний компонентдодатку |

# ДОДАТОК Б

Система поліпшення проведення занять (клієнт  Android)

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51194_19Б

Аркушів 7

Київ 2019

# Текст основного класу

```
import android.os.Bundle
import android.support.design.widget.BottomNavigationView
import android.support.v4.app.Fragment
import android.support.v7.app.AppCompatActivity
import com.example.mydiploma.R
import com.example.mydiploma.remote.FirebaseClass
import com.example.mydiploma.viewModel.MainViewModel
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
class MainActivity : AppCompatActivity(), NavigationInMainActivity {
private val subjectFragment by lazy { SubjectFragment() }
private val settingsFragment by lazy { SettingsFragment() }
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
val navView: BottomNavigationView = findViewById(R.id.nav_view)
navView.setOnNavigationItemSelectedListener(onNavigationItemSelectedListener)
getLiveFromFirebaseDb(MainViewModel.currentUser.group!!)
}
private val onNavigationItemSelectedListener = BottomNavigationView.OnNavigationItemSelectedListener { item ->
when (item.itemId) {
R.id.navigation_live -> {
getLiveFromFirebaseDb(MainViewModel.currentUser.group!!)
return@OnNavigationItemSelectedListener true
}
R.id.navigation_dashboard -> {
loadFragment(subjectFragment,true)
return@OnNavigationItemSelectedListener true
}
R.id.navigation_settings -> {
loadFragment(settingsFragment,true)
return@OnNavigationItemSelectedListener true
}
}
false
}
private fun loadFragment(fragment : Fragment, flag : Boolean) {
val fragmentTransaction = supportFragmentManager.beginTransaction()
when(flag){
true ->fragmentTransaction.replace(R.id.fragmentContainer,fragment).commit()
false -> fragmentTransaction.replace(R.id.fragmentContainer,fragment).addToBackStack("").commit()
}
}
override fun goToLectureFragment(subjectId: String, subjectName : String) {
loadFragment( LectureFragment(subjectId, subjectName),false)
}
override fun goToQuestionFragment(lectureId: String, lectureName: String) {
loadFragment( QuestionFragment(lectureId,lectureName),false)
}
override fun goToAnswerFragment(questionId: String, questionName: String) {
loadFragment( AnswerFragment(questionId,questionName), false)
}
fun getLiveFromFirebaseDb(group: String) =
FirebaseClass.myRef.child("groups").child(group).child("live").addValueEventListener(liveListener)
private var liveListener: ValueEventListener = object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {
val live = dataSnapshot.child("1").getValue(LiveEntity::class.java)
loadFragment(QuestionFragment(live!!.lecture,live.lectureName), true)
}
override fun onCancelled(p0: DatabaseError) {}
}
}
```

# Текст класу Питань в реальному часі

```
import android.annotation.SuppressLint


import android.content.Context
import android.os.Bundle
import android.support.v4.app.Fragment
import android.support.v7.widget.LinearLayoutManager
import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup
import com.example.mydiploma.R
import com.example.mydiploma.data.model.QuestionEntity
import com.example.mydiploma.remote.FirebaseClass
import com.example.mydiploma.ui.adapter.AdapterForQuestionAndAnswer
import com.example.mydiploma.viewModel.MainViewModel
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import kotlinx.android.synthetic.main.fragment_live.view.*
@SuppressLint("ValidFragment")
class QuestionFragment(val lectureId: String, val lectureName: String) : Fragment() {
private lateinit var viewL : View
lateinit var navigationInterface : NavigationInMainActivity
override fun onAttach(context: Context?) {
super.onAttach(context)
navigationInterface = context as NavigationInMainActivity
}
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
viewL = inflater.inflate(R.layout.fragment_live, null)
viewL.lectureName.text = lectureName
viewL.questionTextInput.hint = "Type your question..."
viewL.questionRecyclerView.layoutManager = LinearLayoutManager(context)
viewL.questionRecyclerView.adapter = AdapterForQuestionAndAnswer<QuestionEntity>{
navigationInterface.goToAnswerFragment(it.id,it.text)
}
viewL.sendButton.setOnClickListener {
if (!checkExistTextInTextInput() ) {

addQuestionToFirebaseDb(getQuestionEntityFromTextInput(),MainViewModel.currentUser.group!!)
}
clearTextInput()
}
getQuestionsFromFirebaseDb(MainViewModel.currentUser.group!!)
return viewL
}
private fun checkExistTextInTextInput() = viewL.questionTextInput.text.isNullOrBlank()
private fun getQuestionEntityFromTextInput() =
QuestionEntity("", lectureId, viewL.questionTextInput.text.toString())
private fun clearTextInput() = viewL.questionTextInput.setText("")
fun getQuestionsFromFirebaseDb(group: String) =
FirebaseClass.myRef.child("groups").child(group).child("questions").addValueEventListener(questionListener)
fun addQuestionToFirebaseDb(questionEntity: QuestionEntity, group: String) {
val key = lectureId + (viewL.questionRecyclerView.adapter as AdapterForQuestionAndAnswer<QuestionEntity>).list.size.toString()
questionEntity.id = key
FirebaseClass.myRef.child("groups").child(group).child("questions")
.child(key).setValue(questionEntity)
}
var questionListener: ValueEventListener = object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {
val questions : MutableList<QuestionEntity> = mutableListOf()
for (questionSnapshot : DataSnapshot in dataSnapshot.children) {
val question = questionSnapshot.getValue(QuestionEntity::class.java)!!
if (question.lectureId == lectureId) questions.add(question)
}
(viewL.questionRecyclerView.adapter as AdapterForQuestionAndAnswer<QuestionEntity>)
.updateList(questions)
}
override fun onCancelled(p0: DatabaseError) {}
}
}
```

# Текст класу актуальних Предметів

```
import android.content.Context
import android.os.Bundle
import android.support.v4.app.Fragment
import android.support.v7.widget.LinearLayoutManager
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.mydiploma.R
import com.example.mydiploma.data.model.SubjectEntity
import com.example.mydiploma.remote.FirebaseClass
import com.example.mydiploma.ui.adapter.Adapter
import com.example.mydiploma.viewModel.MainViewModel
```

```
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import io.reactivex.schedulers.Schedulers
import kotlinx.android.synthetic.main.fragment_subject.view.*
import org.koin.android.viewmodel.ext.android.sharedViewModel
class SubjectFragment : Fragment() {
private lateinit var viewL : View
lateinit var navigationInterface : NavigationInMainActivity
private val viewModel by sharedViewModel<MainViewModel>()
override fun onAttach(context: Context?) {
super.onAttach(context)
navigationInterface = context as NavigationInMainActivity
}
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
viewL = inflater.inflate(R.layout.fragment_subject, null)
viewL.recyclerView.layoutManager = LinearLayoutManager(context)
viewL.recyclerView.adapter = Adapter<SubjectEntity>{
navigationInterface.goToLectureFragment(it.id, it.name)
}
viewModel.getSubject().observe(this, object : Observer<List<SubjectEntity>>{
override fun onChanged(subjects: List<SubjectEntity>?) {
if (subjects != null) (viewL.recyclerView.adapter as Adapter<SubjectEntity>).updateList(subjects)
}
})
getSubjectsFromFirebaseDb(MainViewModel.currentUser.group!!)
return viewL
}
fun getSubjectsFromFirebaseDb(group : String) =
FirebaseClass.myRef.child("groups").child(group).child("subjects").addValueEventListener(subjectListener)
var subjectListener: ValueEventListener = object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {
val subjects : MutableList<SubjectEntity> = mutableListOf()
for (subjectSnapshot : DataSnapshot in dataSnapshot.children) {
subjects.add(subjectSnapshot.getValue(SubjectEntity::class.java)!!)
}
viewModel.setSubjects(subjects)
}
override fun onCancelled(p0: DatabaseError) {}
}
}
```

# Текст класу Профілю

```
importandroid.os.Bundle


import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.mydiploma.R
import com.example.mydiploma.viewModel.MainViewModel
import kotlinx.android.synthetic.main.fragment_settings.*
import kotlinx.android.synthetic.main.fragment_settings.view.*
class SettingsFragment : Fragment() {
lateinit var viewL : View
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
viewL = inflater.inflate(R.layout.fragment_settings, null)
viewL.nameText.setText( MainViewModel.currentUser.name.toString())
viewL.eMailText.setText(MainViewModel.currentUser.mail)
viewL.phoneText.setText(MainViewModel.currentUser.phone)
if (MainViewModel.currentUser.isteacher) viewL.statusTextView.setText("Teacher")
viewL.aboutButton.setOnClickListener { aboutButtonClicked() }
viewL.backButton.setOnClickListener { backButtonClicked() }
viewL.exitButton.setOnClickListener { System.exit(0) }
return viewL
}
private fun aboutButtonClicked() {
viewL.nameText.visibility = View.INVISIBLE
viewL.eMailText.visibility = View.INVISIBLE
viewL.exitButton.visibility = View.INVISIBLE
viewL.aboutButton.visibility = View.INVISIBLE
viewL.aboutInfo.visibility = View.VISIBLE
viewL.phoneText.visibility = View.INVISIBLE
viewL.statusTextView.visibility = View.INVISIBLE
viewL.view2.visibility = View.INVISIBLE
```

```
personImageView.visibility = View.INVISIBLE
}
private fun backButtonClicked() {
viewL.nameText.visibility = View.VISIBLE
viewL.eMailText.visibility = View.VISIBLE
viewL.exitButton.visibility = View.VISIBLE
viewL.aboutButton.visibility = View.VISIBLE
viewL.aboutInfo.visibility = View.INVISIBLE
viewL.phoneText.visibility = View.VISIBLE
viewL.statusTextView.visibility = View.VISIBLE
viewL.view2.visibility = View.VISIBLE
personImageView.visibility = View.VISIBLE
}
}
```

# Текст класу Меню

```
package com.example.mydiploma.ui


import android.animation.FloatEvaluator
import android.content.Intent
import android.support.v7.app.AppCompatActivity
import android.support.v4.app.Fragment
import android.support.v4.app.FragmentManager
import android.support.v4.app.FragmentPagerAdapter
import android.support.v4.view.ViewPager
import android.os.Bundle
import android.util.Log
import android.view.View
import com.example.mydiploma.*
import com.example.mydiploma.remote.FirebaseClass
import kotlinx.android.synthetic.main.activity_hello_page.*
class HelloPageActivity : AppCompatActivity(), NavigationInterface {
private var mSectionsPagerAdapter: SectionsPagerAdapter? = null
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_hello_page)
mSectionsPagerAdapter = SectionsPagerAdapter(supportFragmentManager)
container.adapter = mSectionsPagerAdapter
container.setOnPageChangeListener(object : ViewPager.OnPageChangeListener{
override fun onPageScrollStateChanged(p0: Int) {}
override fun onPageScrolled(p0: Int, p1: Float, p2: Int) {
if (p0 == 0) {
signInTextView.textSize = FloatEvaluator().evaluate(p1,24f,16f)
registrationTextView.textSize = FloatEvaluator().evaluate(p1,16f,24f)
}
else {
signInTextView.textSize = FloatEvaluator().evaluate(p1,16f,24f)
registrationTextView.textSize = FloatEvaluator().evaluate(p1,24f,16f)
}
}
override fun onPageSelected(p0: Int) {}
})
}
override fun onBackPressed() {
super.onBackPressed()
signInTextView.visibility = View.VISIBLE
registrationTextView.visibility = View.VISIBLE
container.visibility = View.VISIBLE
}
inner class SectionsPagerAdapter(fm: FragmentManager) : FragmentPagerAdapter(fm) {
override fun getItem(position: Int): Fragment {
return when (position) {
0 -> SignInFragment()
1 -> RegistrationFragment()
else -> Fragment()
}
}
override fun getCount() = 2
}
override fun goToMainActivity() {
val intent = Intent (this, MainActivity::class.java)
startActivity(intent)
}
```

# Текст класу Авторизації

```
package com.example.mydiploma.ui

import android.content.Context
import android.os.Bundle
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.mydiploma.R
import com.example.mydiploma.data.model.User
import com.example.mydiploma.remote.FirebaseClass
import com.example.mydiploma.viewModel.MainViewModel
import com.google.android.gms.tasks.OnCompleteListener
import com.google.android.gms.tasks.Task
import com.google.firebase.auth.AuthResult
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import kotlinx.android.synthetic.main.fragment_sign_in_page.view.*


class SignInFragment : Fragment() {

lateinit var navigationInterface : NavigationInterface
lateinit var auth : FirebaseAuth

var userListener: ValueEventListener = object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {
for (userSnapshot : DataSnapshot in dataSnapshot.children) {
if (userSnapshot.key == auth.uid)  {
MainViewModel.currentUser = userSnapshot.getValue(User::class.java)!!
MainViewModel.currentUser.mail = auth.currentUser?.email!!
navigationInterface.goToMainActivity()
break
}
}
}
override fun onCancelled(p0: DatabaseError) {}
}

lateinit var viewL : View

override fun onAttach(context: Context?) {
super.onAttach(context)
navigationInterface = context as NavigationInterface
}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

viewL = inflater.inflate(R.layout.fragment_sign_in_page, null)
auth = FirebaseAuth.getInstance()
viewL.signInButton.setOnClickListener {
viewL.signInProgressBar.visibility = View.VISIBLE
signInUser(viewL.emailEditText.text.toString(), viewL.passwordEditText.text.toString())
}

return viewL
}

private fun signInUser(email : String, password : String) {
auth.signInWithEmailAndPassword(email,password)
.addOnCompleteListener(activity!!, object : OnCompleteListener<AuthResult>{
override fun onComplete(p0: Task<AuthResult>) {
viewL.signInProgressBar.visibility = View.INVISIBLE
if (p0.isSuccessful) {
getUsersFromFirebaseDb()
}
}

})
}

fun getUsersFromFirebaseDb() = FirebaseClass.myRef.child("users").addValueEventListener(userListener)

}
```

# Текст класу реєстрації

```
package com.example.mydiploma.ui

import android.os.Bundle

import android.support.v4.app.Fragment

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.Toast

import com.example.mydiploma.R

import com.example.mydiploma.data.model.User

import com.example.mydiploma.remote.FirebaseClass

import com.google.android.gms.tasks.OnCompleteListener

import com.google.android.gms.tasks.Task

import com.google.firebase.auth.AuthResult

import com.google.firebase.auth.FirebaseAuth

import kotlinx.android.synthetic.main.fragment_registration_page.view.*


class RegistrationFragment : Fragment() {

lateinit var auth : FirebaseAuth

val fb = FirebaseClass()

lateinit var viewL: View

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

viewL = inflater.inflate(R.layout.fragment_registration_page, null)

auth = FirebaseAuth.getInstance()

viewL.registrationButton.setOnClickListener { registrationUser(viewL.emailEditText.text.toString(), viewL.passwordEditText.text.toString()) }

return viewL

}

private fun registrationUser(email: String, password: String) {

auth.createUserWithEmailAndPassword(email,password)

.addOnCompleteListener(activity!!, object : OnCompleteListener<AuthResult>{

override fun onComplete(p0: Task<AuthResult>) {
```

```
if (p0.isSuccessful) {

Toast.makeText(context!!,"Registration sucess", Toast.LENGTH_SHORT).show()

addUserToFirebaseDb(

User(

"ti-51",

false,

p0.result?.user?.email,

viewL.nameEditText.text.toString(),

viewL.phoneEditText.text.toString()

), p0.result?.user?.uid!!)

}

}

})

}

fun addUserToFirebaseDb(user: User, key: String) = FirebaseClass.myRef.child("users").child(key).setValue(user)

}
```

# ДОДАТОК В

Система поліпшення проведення занять (клієнт  Android)

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51194_19Б

Аркушів 9

Київ 2019

# АНОТАЦІЯ

Даний додаток містить опис основних модулів додатку для звязку між вчитеем та учнем.
Програмний продукт створений мною виконує такі завдання:
—реєстрація та авторизація у додатку;
—введення лекцій у режимі реального часу;
—перегляд актуальних предметів та лекцій;
—відмічення вчителем правильних відповідей;
—перегляд профілю.
Працює додаток на всіх гаджет під платформою Android.
При розробці системи застосовувались Kotlin та середовище програмування Android Studio, з використанням мобільної API — Firebase.

# ЗМІСТ

# ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку є опис головних класів системи для введення предметів онлайн, перегляд актуальних предметів та лекцій, авторизації та реєстрації у додатку, яка доступна тільки для студентів
Програмний продукт працює на всіх гаджетах під платформою Android.
При програмуванні даного додатку використовувалась мова Kotlin та середовище розробки Android Studio.

# ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний продукт має такі завдання як проведення онлайн занять, перегляд актуальних предметів та лекцій, перегляд профілю, авторизація та реєстрація у додатку.
Додаток може використовуватись у всіх навчальних закладах
Функціональні обмеження немає

# ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для створення програмного продукту потрібна було підключення сторонніх бібліотек.
Для реалізації додатку на всіх гаджетах під платформою Android використовувалась бібліотека Support Library.
Android Studio має вбудовану базу даних SqlLite, але в ній є ряд недоліків і тому було прийняте рішення про використання бібліотеки Room.
Після встановлення додатку вас вітає вікно входу в систему, після нього вікно онлайн лекцій, після якого з легкістю можна переключатись між актуальними предметами та профілем користувача.

# ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для покращення роботи серед всіх додатків під платформою Android, використовувались різні технології такі як Suport Library та тестування та емуляторах різних версій Android
Програмний продукт працює на всіх додатках Android починаючи з API 26 закінчуючи найновішими API

# ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмний продукт знаходиться у розширенні .apk - “LectureTime.apk” потрібно встановити його на емулятор через Android Studop та натиснути на інстальований додаток.
Користувач отримає демонстраційну версію додатку, в які потрібно пройти реєстарацію, і після цього представлене основне вікно.

# ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними вважаються екрани входу та реєстрації, які беруть інформацію з editText, та повідомлення питання та відповіді.
Вхідні дані мають бути заповнені у відповідності до вказаних полів.
Вихідними даними вважаються отримані від мобільного API — Firebase дані та дані про користувача у вікні профілю